```
#importing all the necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
```

```
C:\Users\gugan\AppData\Local\Temp\ipykernel_9496\1230224523.py:2:
DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major
release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type,
and better interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at
https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

```
#loading the data set ,the dataset which is given to me is retail
sales dataset i'm loading it using the pandas library

data = pd.read_csv(r"C:\Users\gugan\Desktop\Quinnel_soft\
retail_sales.csv")

# Display the first few rows and the column names
print(data.head())
print(" ")
print('-----------------------')
print(" ")
print(data.columns)
```

```
          Date      Sales
0  1/1/15 0:00  54.048619
1  1/1/15 1:00  50.161596
2  1/1/15 2:00  55.954939
3  1/1/15 3:00  56.553261
4  1/1/15 4:00  60.487874

-----------------------

Index(['Date', 'Sales'], dtype='object')
```

```
data.columns = data.columns.str.strip()
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')
```

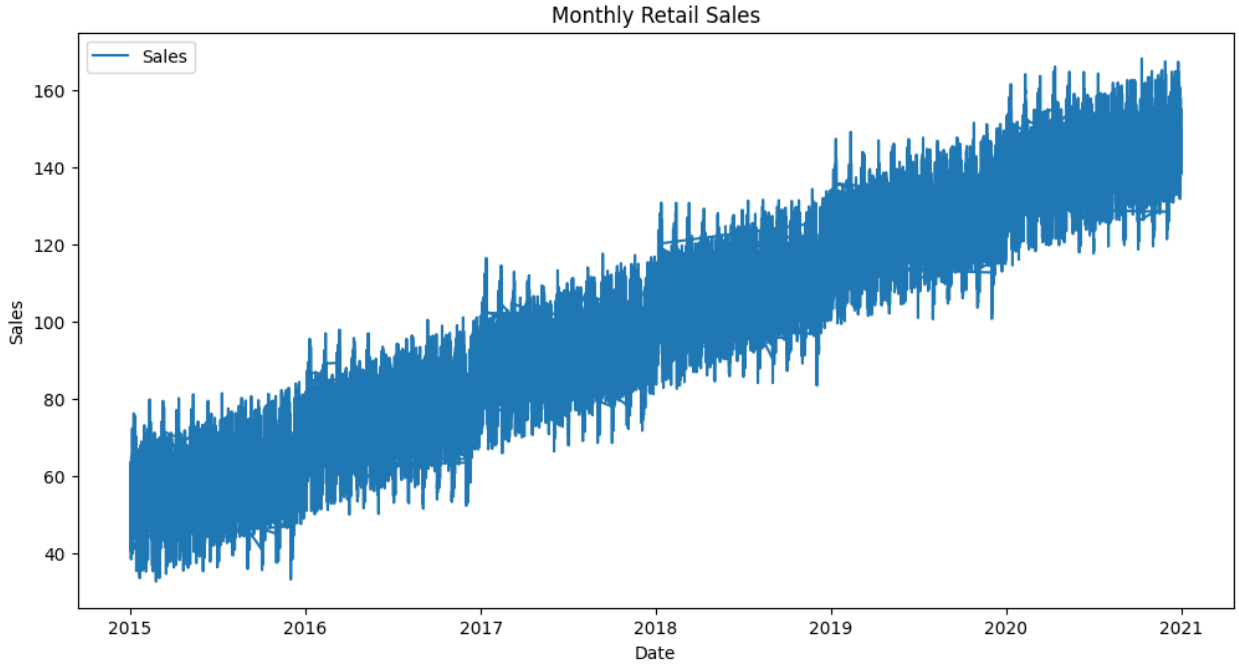
```
data.set_index('Date', inplace=True)

# Check the result
print(data.head())
print(data.info())
```

```
C:\Users\gugan\AppData\Local\Temp\ipykernel_9496\4105646762.py:2:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is
consistent and as-expected, please specify a format.
  data['Date'] = pd.to_datetime(data['Date'], errors='coerce')

                         Sales
Date
2015-01-01 00:00:00  54.048619
2015-01-01 01:00:00  50.161596
2015-01-01 02:00:00  55.954939
2015-01-01 03:00:00  56.553261
2015-01-01 04:00:00  60.487874
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 52585 entries, 2015-01-01 00:00:00 to 2020-12-31
00:00:00
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Sales   52585 non-null  float64
dtypes: float64(1)
memory usage: 821.6 KB
None
```

```
# Visualize the time series
plt.figure(figsize=(12, 6))
plt.plot(data.index, data['Sales'], label='Sales')
plt.title('Monthly Retail Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.legend()
plt.show()

# Handle missing values
data = data.dropna()

# feature Scaling the data
scaler = MinMaxScaler(feature_range=(0, 1))
data['Sales'] = scaler.fit_transform(data['Sales'].values.reshape(-
1,1))

# Preparing the data for LSTM
def create_dataset(dataset, look_back=1):
    X, Y = [], []
```

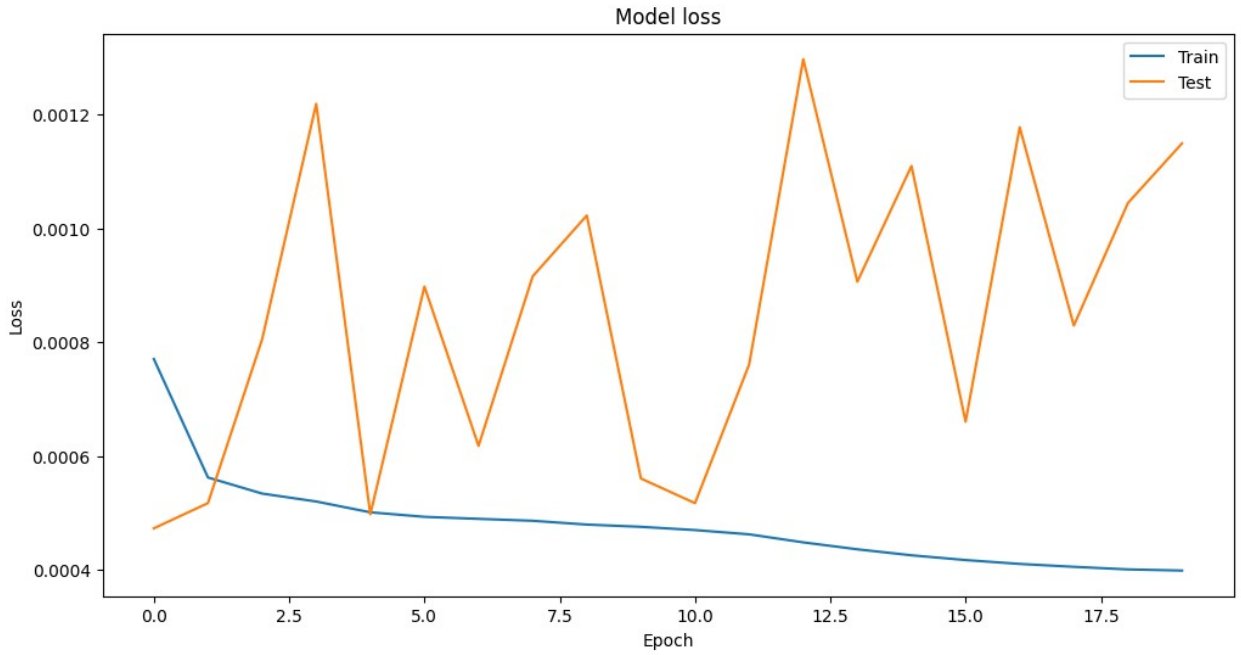
```
    for i in range(len(dataset)-look_back-1):
        a = dataset[i:(i+look_back), 0]
        X.append(a)
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

look_back = 12
X, Y = create_dataset(data.values, look_back)

# Split into train and test sets
train_size = int(len(X) * 0.8)
test_size = len(X) - train_size
X_train, X_test = X[0:train_size], X[train_size:len(X)]
Y_train, Y_test = Y[0:train_size], Y[train_size:len(Y)]

# Reshape input to be [samples, time steps, features]
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))

# Build and train the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(look_back, 1)))
model.add(LSTM(50, return_sequences=False))
model.add(Dense(25))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

history = model.fit(X_train, Y_train, batch_size=1, epochs=20,
validation_data=(X_test, Y_test))

# Plot training & validation loss values
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
plt.show()
```

```
C:\Users\gugan\AppData\Local\Temp\ipykernel_21028\3784555460.py:13:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is
consistent and as-expected, please specify a format.
  data['Date'] = pd.to_datetime(data['Date'], errors='coerce')  #
Convert to datetime, handling errors
```




```
Epoch 1/20

C:\Python312\Lib\site-packages\keras\src\layers\rnn\rnn.py:204:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a
layer. When using Sequential models, prefer using an `Input(shape)`
object as the first layer in the model instead.
  super().__init__(**kwargs)

42057/42057 ━━━━━━━━━━━━━━━━━━━━ 262s 6ms/step - loss: 0.0013 -
val_loss: 4.7270e-04
Epoch 2/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 288s 7ms/step - loss: 5.7363e-04 -
val_loss: 5.1709e-04
Epoch 3/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 333s 8ms/step - loss: 5.3874e-04 -
val_loss: 8.0471e-04
Epoch 4/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 298s 7ms/step - loss: 5.2250e-04 -
val_loss: 0.0012
Epoch 5/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 292s 7ms/step - loss: 5.1300e-04 -
val_loss: 4.9738e-04
Epoch 6/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 289s 7ms/step - loss: 4.9250e-04 -
val_loss: 8.9762e-04
Epoch 7/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 267s 6ms/step - loss: 4.9212e-04 -
val_loss: 6.1770e-04
Epoch 8/20
```

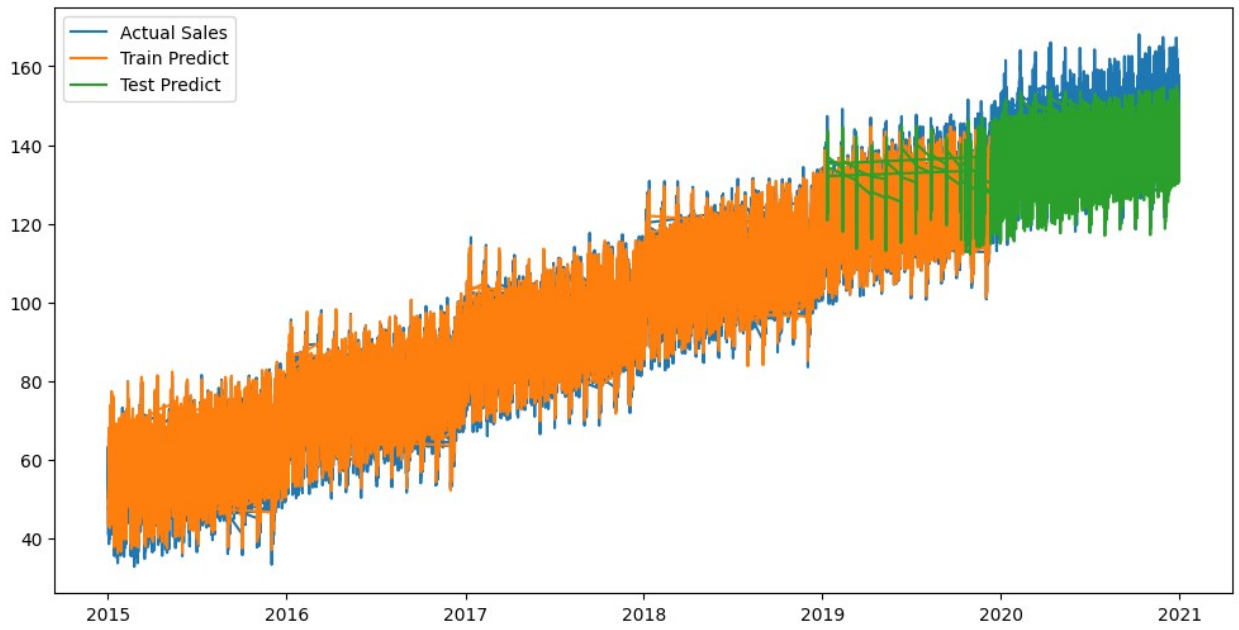
```
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 297s 7ms/step - loss: 4.8651e-04 -
val_loss: 9.1556e-04
Epoch 9/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 265s 6ms/step - loss: 4.7739e-04 -
val_loss: 0.0010
Epoch 10/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 271s 6ms/step - loss: 4.8107e-04 -
val_loss: 5.6031e-04
Epoch 11/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 259s 6ms/step - loss: 4.6670e-04 -
val_loss: 5.1695e-04
Epoch 12/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 279s 7ms/step - loss: 4.6916e-04 -
val_loss: 7.6070e-04
Epoch 13/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 285s 7ms/step - loss: 4.5130e-04 -
val_loss: 0.0013
Epoch 14/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 254s 6ms/step - loss: 4.3716e-04 -
val_loss: 9.0624e-04
Epoch 15/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 259s 6ms/step - loss: 4.2750e-04 -
val_loss: 0.0011
Epoch 16/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 258s 6ms/step - loss: 4.2006e-04 -
val_loss: 6.6019e-04
Epoch 17/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 256s 6ms/step - loss: 4.0872e-04 -
val_loss: 0.0012
Epoch 18/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 250s 6ms/step - loss: 4.0601e-04 -
val_loss: 8.2911e-04
Epoch 19/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 270s 6ms/step - loss: 3.9963e-04 -
val_loss: 0.0010
Epoch 20/20
42057/42057 ━━━━━━━━━━━━━━━━━━━━ 266s 6ms/step - loss: 4.0085e-04 -
val_loss: 0.0011
```



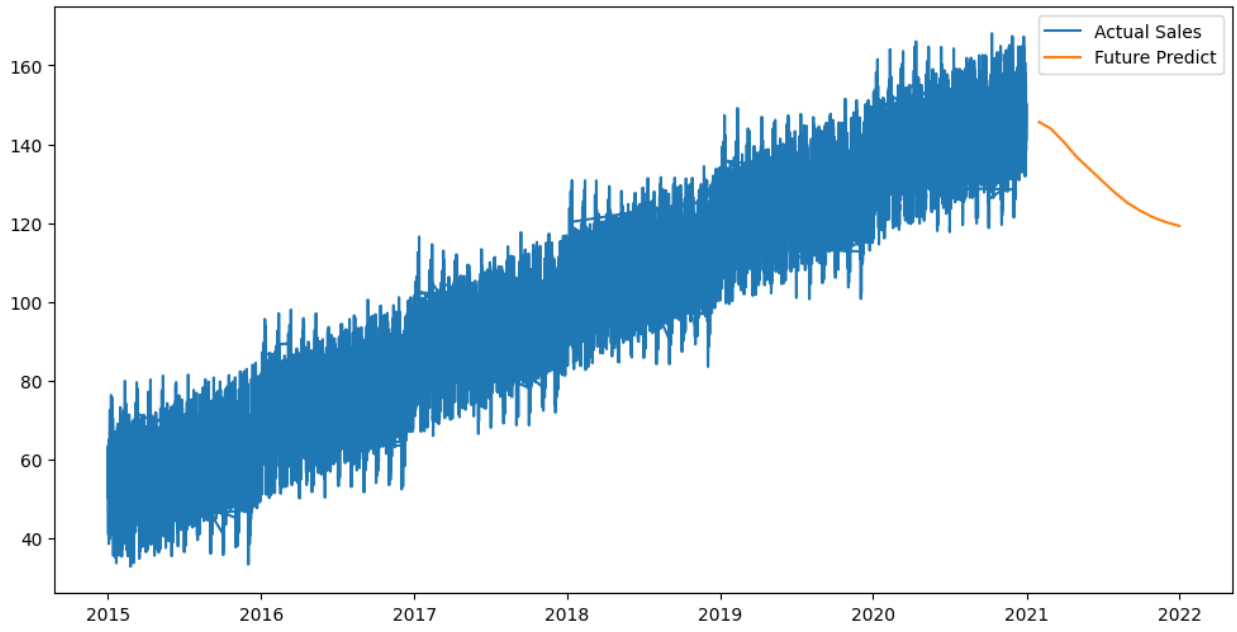

```
# Make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

# Inverse transform predictions
train_predict = scaler.inverse_transform(train_predict)
Y_train = scaler.inverse_transform(Y_train.reshape(-1, 1))
test_predict = scaler.inverse_transform(test_predict)
Y_test = scaler.inverse_transform(Y_test.reshape(-1, 1))

# Calculate performance metrics
train_score_rmse = np.sqrt(mean_squared_error(Y_train, train_predict))
test_score_rmse = np.sqrt(mean_squared_error(Y_test, test_predict))
train_score_mae = mean_absolute_error(Y_train, train_predict)
test_score_mae = mean_absolute_error(Y_test, test_predict)
train_score_r2 = r2_score(Y_train, train_predict)
test_score_r2 = r2_score(Y_test, test_predict)

print(f'Train Score: RMSE={train_score_rmse:.2f},
MAE={train_score_mae:.2f}, R2={train_score_r2:.2f}')
print(f'Test Score: RMSE={test_score_rmse:.2f},
MAE={test_score_mae:.2f}, R2={test_score_r2:.2f}')

# Plot the predictions
plt.figure(figsize=(12, 6))

# Plot actual sales
actual_sales = scaler.inverse_transform(data['Sales'].values.reshape(-
1, 1))
plt.plot(data.index, actual_sales, label='Actual Sales')
```

```
# Plot training predictions
train_plot = np.empty_like(data['Sales'])
train_plot[:] = np.nan
train_plot[look_back:len(train_predict) + look_back] =
train_predict[:, 0]
plt.plot(data.index, train_plot, label='Train Predict')

# Plot testing predictions
test_plot = np.empty_like(data['Sales'])
test_plot[:] = np.nan
test_start_idx = len(train_predict) + (look_back * 2) + 1
test_end_idx = test_start_idx + len(test_predict)

if test_end_idx <= len(test_plot):
    test_plot[test_start_idx:test_end_idx] = test_predict[:, 0]
else:
    test_plot[test_start_idx:len(test_plot)] =
test_predict[:len(test_plot) - test_start_idx, 0]

plt.plot(data.index, test_plot, label='Test Predict')

plt.legend()
plt.show()

# Forecast future sales
future_steps = 12
last_values = data[-look_back:].values
future_predictions = []

for _ in range(future_steps):
    prediction = model.predict(last_values.reshape(1, look_back, 1))
    future_predictions.append(prediction[0, 0])
    last_values = np.append(last_values[1:], prediction[0,
0]).reshape(-1, 1)

future_predictions =
scaler.inverse_transform(np.array(future_predictions).reshape(-1, 1))

# Plot future predictions
future_dates = pd.date_range(data.index[-1], periods=future_steps + 1,
freq='M')[1:]
plt.figure(figsize=(12, 6))
plt.plot(data.index, actual_sales, label='Actual Sales')
plt.plot(future_dates, future_predictions, label='Future Predict')
plt.legend()
plt.show()

1315/1315 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step
329/329 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step
```

```
Train Score: RMSE=31352142983.02, MAE=30261654973.55, R2=-13.63
Test Score: RMSE=47131010746.82, MAE=47013894504.24, R2=-200.46
```




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 79ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 76ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

C:\Users\gugan\AppData\Local\Temp\ipykernel_21028\1078697490.py:64:
FutureWarning: 'M' is deprecated and will be removed in a future
version, please use 'ME' instead.
  future_dates = pd.date_range(data.index[-1], periods=future_steps +
1, freq='M')[1:]
```


Actual Sales
Future Predict
160
140
120
100
80
60
40
2015
2016
2017
2018
2019
2020
2021
2022